Housam Zeineddin

Object-Oriented Programming

Course Work report

IT-fuc 19

My course work is a program that resembles a mini Banking System, providing basic customer transactions and information retrieval, as well as customer searching feature for employees.

The software initializes sample employees and customer on startup, that can be accessed for various functionalities.

A summary of the software is as follows:

The base class “person” of the main subclasses/entities “customer” and “employee” provides login capabilities to the software.

A customer can use the system to display all the info of his/her accounts, as well as withdraw and deposit into their accounts.

An employee can use the system to display his personal info, or to search and display a certain customer’s info using the customer’s full name. He/she can also display info summary of all customers.

The libraries used are:

<iostream> is used for basic input and output and output services.

<iomanip> used to manipulate the output.

<string> used for handling strings

<stdexcept> is used for throwing invalid argument exceptions

<vector> is used to create a vector, which is “daysPerMonth”.

-Class Date:

I created a class to store date information and print in a specific format.

The public members implemented are:

* Vector<int>daysPerMonth: This vector contains 12 elements resembling the 12 months, where each value is the numbers of days of the month (starting at index 0).
* Date (): default constructor.
* Date (int,int,int): Constructor that calls the setdate() function.
* Void setDate(int,int,int): sets the month, day, and year.
* Void setMonth(int): sets the month between 1 and 12 inclusive.
* Void setDay (int): sets the day according to the month, and insures that the day is valid for the set month (including leap year check).
* Void setYear (int): sets the year.
* getMonth () const, getDay () const, and getyear () const are functions that return month, day, and year respectively.
* Void printDate (): print function to print the date in a specific format.

-Class Person:

Class Person implements certain data members and member functions and acts as a base for the Customer and Employee classes. Each person object has first name, last name, username, password, and birthday (which is a Date object). Set and get functions are implemented for all data members. The virtual functions printInfo () and getRole() define basic behaviour and will be overloaded by subclasses.

The member functions I implemented in this class are:

* Person (): default constructor.
* Person (string, string, string, string, int, int, int): Parameterized constructor, it sets values to the private data members of the class.
* Void setFirstName (string), void setLastName (string), void setUserame (string), void setPassword (string), are functions that set the first name, last name, username, and password respectively.
* Void string getFirstName () const, void string getLastName () const, void string getUsername () const, void string getpassword () const are functions that return the first name, last name, username, and password respectively.
* Void setBirthDay (Date): set Birth day.
* Date getBirthDay () const: returns BirthDay.
* Virtual string getRole (): virtual function. Returns default role ‘person’, and when overridden by the sub classes, will provide the Person class with polymorphic abilities.
* Virtual void printInfo (): virtual function. Prints default info, and when overridden by the sub classes, will provide the Person class with polymorphic abilities.
* “Date.h” is included to implement a private data member of type Date.

-Class Employee:

Class Employee inherits from Class Person, and provides an extra data member ‘salary’. Several member functions are added to this class, which are:

* Employee(string, string, string, string, float, int, int, int): Constructor for object Employee. It calls the Person constructor and sets a value for the salary.
* Float getSalary () const : returns the salary.
* Void printInfo () override: overridden function that prints the Employee information.
* String getRole () returns the role of the Customer which is “Customer”.

-Class Customer:

Class Customer inherits from Class Person and provides an extra data member which is a vector of ‘BankAccount’ allowing a customer to have multiple bank accounts. The member functions I implemented in this class are:

* void addBankAccount (BankAccount): which pushes a BankAccount object to the bankAccounts vector.
* Float getBalanceForAccount(string): searches by accountNumber for an account in the bankAccounts vector, and returns this account’s balance, or -1 if account not found.
* Float getTotalBalance () const: returns the total balance of a customer by adding the balance of all individual accounts. It utilizes the overridden operator+ of the BankAccount class while iterating through the customer’s bankAccounts.
* Float getNumberOfAccounts () const: returns the size of the vector bankAccounts.
* Bool withdraw (string, float): Gets a customer’s bank account by accountNumber, using the getBalanceForAccount(string), and if found, calls the withdraw function of the bank account to withdraw from it. The function return false if the account was not found or the amount to withdraw is invalid. Polymorphism is resembled here when withdrawing from a normal BankAccount or LimitedBankAccount, where LimitedBankAccount is a subclass of the BankAccount class that overrides the withdraw function.
* Bool deposit (string, float) Gets a customer’s bank account by accountNumber, using the getBalanceForAccount(string), and if found, calls the deposit function of the bank account to deposit into it. The function returns false if the account was not found or the amount to deposit is invalid.
* Void printInfo () override: overridden function that prints the Customer information.
* String getRole () returns the role of the Customer which is “Customer”.

-Class BankAccount:

Class BankAccount defines a bank account with an accountNumber and balance as data members. This class defines an account object that will belong to customers. It defines member functions to withdraw and deposit into its balance data member.

The private data members are string accountNumber, and float balance.

The member functions of this class are:

* BankAccount (string, float): constructor that sets the account number and balance of the object.
* Float operator+(float): overridde +operator to directly return the sum of the account’s balance added to a flot number. This will be used when calculating the total balance of all accounts of a customer.
* Void setAccountNumber (string): function to set account number.
* Void getAccountNumber () const: function that returns the account number
* Void setBalance (float): function to set balance.
* Float getBalance () const: function that returns balance.
* Virtual bool withdraw (float): Attempts to withdraw from account. This function returns false if the amount is negative or if the amount exceeds the balance. Otherwise, the amount is deducted from the balance and true is returned.
* Bool deposit (float): Attempts to deposit and amount into the account. This function returns false if the amount is. Otherwise, the amount is added to the balance and true is returned.
* Virtual string getAccountType (): virtual function. Returns default type ‘normal’, and when overridden by the sub class, will provide the BankAccount class with polymorphic abilities.
* Virtual void printInfo (): virtual function. Prints default info, and when overridden by the sub class, will provide the BankAccount class with polymorphic abilities.

-Class LimitedBankAccount :

This class inherits from BankAccount Class, and overrides the withdraw function to limit it to a maximum of $200 per withdraw transaction. Its member functions are:

* LimitedBankAccount (string, float): It is this class’s constructor; it calls the execution of BankAccount’s constructor.
* Bool withdraw (float) override: Overrides the withdraw function of the BankAccount base class. Attempts to withdraw from account. This function returns false if the amount is negative, if the amount exceeds the balance, or if the transaction is more than $200. Otherwise, the amount is deducted from the balance and true is returned.
* String getAccountType () override: This function overrides that function from the BankAccount base class, returning the account type “limited”. Affects the output of the base class printInfo function that shows the account type.

-Class Source :

This is the main class that runs the program. Its composed of multiple functions along with the main function as follows:

Global variables:

* User. The logged in user object of type person.
* People. A vector of ‘Person’ objects(customers and employees)

**main():** The main function that runs the program. It first calls the init() function to populate the ‘people’ vector with sample customers and employees.

Then the user is asked to enter his/her credentials, which are validated by the validateUser() function. If the user types -1 for either the username or password, the software terminates. If the credentials are invalid, the user is asked to log in again.

If the user is a customer, the handleCustomer() function is called, and if the user is an employee, the handleEmployee() function is called. Otherwise the user will be warned of an invalid role and prompted to log in again.

**init():** Initializes the ‘people’ vector with sample customers and employees.

**validateUser(username,password):** searches the ‘people’ vector for the specified username. If the password is correct, the ‘person’ will be returned. Otherwise if the password is wrong or the username was not found, NULL will be returned.

**handleEmployee():** Displays the employee menu to the logged in employee. The employee can view personal information, search for a customer, view all customer information, log out, or exit.

To view the personal information, printInfo() function is called on the user object. Since the user object here (of type ‘Person’) is an Employee object, the printInfo() function will call the Employee’s overridden printInfo() function, displaying the employee’s information.

To search for a customer, the employee is asked to input the first name and last name of the customer or -1 in either the first name or last name to cancel the search(note that the input is intended for 2 variables firstname and lastname, so cancelling will require both of the variables populated, with either of them having the value -1), then the searchCustomer(firstName,lastName) function is called to retrieve the customer. If the customer was not found, the employee is prompted to enter first and last name again. Otherwise, the printInfo() function is called on the retrieved customer object.

To view all customer information, the program iterates over all objects of the ‘people’ vector, detects the customer objects using the person->getRole() function (which is overridden by both the Employee and Customer classes), and prints the customer info using the overridden operator<< of the Customer class.

To log out, the program sets the user to null, which will allow the loop in the main function to continue, asking for new login credentials.

To exit the software, the option ‘-1’ entered by the user will terminate the main loop of the handleEmployee() function, and since the user was not reset, the main loop of the main function will also terminate.

**searchCustomer(firstName,lastName):**  The function will search the ‘people’ vector for a person that matches the first name and last name provided, who is also a customer, and returns the customer, or NULL if not found.

**handleCustomer():** Displays the customer menu to the logged in customer. The customer can view personal information, withdraw from an account, deposit into an account, log out, or exit.

To view the personal information, printInfo() function is called on the user object. Since the user object here (of type ‘Person’) is a Customer object, the printInfo() function will call the Customer’s overridden printInfo() function, displaying the customer’s information.

To withdraw from an account, the customer is asked to input the account number of the account to withdraw from. The accountnumber is validated by calling the customer->checkAccountNumber(accountNumber). An invalid account will ask the customer to enter another account number. Then the customer will be asked to input the amount to withdraw.

Then the customer-> withdraw(accountNumber,amount) is called. If the account is a LimitedBankAccount, a withdrawamount greater than $200 will fail(due to the condition in the overridden withdraw function of the LimitedBankAccount subclass of the BankAccount base class). The user will then be notified of either a successful transaction message, with the remaining balance of the account, or a transaction failed message it the transaction failed.

To deposit into an accoun, the customer is asked to input the account number of the account to deposit into. The accountnumber is validated by calling the customer->checkAccountNumber(accountNumber). An invalid account will ask the customer to enter another account number. Then the customer will be asked to input the amount to deposit.

Then the customer-> deposit(accountNumber,amount) is called. The user will then be notified of either a successful transaction message, with the updated balance of the account, or a transaction failed message it the transaction failed.

To log out, the program sets the user to null, which will allow the loop in the main function to continue, asking for new login credentials.

To exit the software, the option ‘-1’ entered by the user will terminate the main loop of the handleEmployee() function, and since the user was not reset, the main loop of the main function will also terminate.

The polymorphic abilities of the classes allows us to easily extend our software to support different types of people(Person) without having to rewrite code, and supporting a new type in main will be as simple as adding a ‘handleNewType()’ function.

The software has some limitations, one which is that any changes made to customer accounts will not persist over multiple executions of the software. To persist the information, we could have used files to write changes to, and to initialize the ‘people’ vector from on startup.

Sample Employees:

1. Name: Jacob

Surname: Oconnor

Username: jacob

Password: jacob

1. Name: Martha

Surname: Rogen

Username: martha

Password: martha

1. Name: Ralph

Surname: Archur

Username: ralph

Password: ralph

Sample Customers:

1. Name: Marshal

Surname: Mathers

Username: marshal

Password: marshal

1. Name: Jon

Surname: Bosewick

Username: jon

Password: jon